Fine-Grained Car Make and Model Classification with Transfer Learning and BCNNs

Dillon Pulliam and Hashim Saeed
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`{jpulliam and hashims}@andrew.cmu.edu`

Abstract

In this project we explore fine-grained car make and model classification on the *Stanford Cars Data Set*. We first experiment with fine-tuning some of the more famous CNN architectures such as VGG, Resnet, and Densenet. After doing this analysis we build various structured ensembles of these fine-tuned models and analyze how they are able to support each other during classification. Finally we explore the concept of bilinear convolutional neural networks (BCNNs) which take into consideration not only spatial locality within images but also feature location.

1 Introduction

For our group's final project we have chosen to do a version of image classification. In particular we work with the *Stanford Cars Data Set* [12]. This data set contains 16,185 images of 196 different classes of cars. Car classifications are on the level of *Make*, *Model*, and *Year*. This data has been split nearly in half in regard to training and test sets with the training set containing 8,144 images and test set 8,041. Our primary goal is to classify cars as accurately as possible. We examine accuracy in regard to both the top predicted class as well as the top five predicted classes. Examples of images from our data set are as follows:

(a) Ferrari FF Coupe 2012

(b) Aston Martin V8 Vantage Coupe 2012

(c) Mercedes-Benz Sprinter Van 2012

One difficulty in this project is the similarity between classes. As we are working with a data set that only contains car images, there is significant overlap in representation between one class and another. Another difficulty is in the vast variation between images due to different backgrounds, lighting, camera orientations, and image resolutions. These difficulties, along with the fact that the training set is fairly small, 8,144 images for 196 total classes, make this problem difficult to tackle.

The importance of fine-grained classification problems such as this cannot be under-stated and is an important research topic. Consider a future scenario where traffic light cameras are used to track missing or stolen vehicles. Or, in another similar scenario, where autonomous robots are tasked with staffing a warehouse and loading shipping containers with the proper goods. In both cases fine-grained classification is crucial as it plays a role in distinguishing specific cars or specific goods from other similar items.

2 Background

For our baseline model implementation we use the well-known, well-documented AlexNet [14] architecture. Here a NVIDIA GeForce GTX 1060 GPU is used for training with source code utilizing the PyTorch [21] machine learning library and pre-trained TorchVision models.

Two training techniques are implemented and analyzed for this model. In technique one we train the entire network "from-scratch" by randomly initializing all parameters and using stochastic gradient descent. Unfortunately this technique fails and the model never begins learning within sixty epochs. Our reasoning is that due to the enormous size of the model, over 62 million parameters, and the relatively small size of the training set along with similarity between classes, learning is difficult.

In our second variation of training we implement fine-tuning, where a pre-trained model is loaded in and trained from this state on a different data set. In this case the pre-trained ImageNet [5] version of AlexNet is used with a slight adjustment in the final fully-connected layer size due to fewer classes. Here our results ware significantly better and after thirty epochs of training we are able to reach 41.82% accuracy (58.18% error) in regard to the top model prediction and 69.05% accuracy (30.95% error) on top five predictions. This demonstrates the enormous power of transfer learning which is a crucial building block going forward in later analysis.

3 Related Work

The primary aim of our project is classification of images of different cars, a field which has extensive applications especially in traffic monitoring, car recognition, etc. All of these applications are based on image processing and classification which deeply involve the use of Convolutional Neural Networks (CNN) because they learn the features as well as the weights thus resulting in a more accurate model. For our project, we make use of two preexisting techniques to aid us in classification; Transfer Learning and Fine-Grained Classification.

For many machine learning algorithms and neural networks, there is an assumption that the training and the test of future data must be in the same feature space. However, in many scenarios this is not applicable as there may be limited data in the feature space we need to classify in. Transfer Learning is the technique through which we use pre-learned classifiers for our tasks[20]. These classifiers are then transferred to our network and only a small subset of the base network is retrained on our data set. This in turn reduces the complexity and saves a lot of computational time [18]. Transfer Learning has been used extensively for various applications such as Web-Document Classification [20] where the primary aim is to classify the website into a set of categories by reviewing the content of the web-page. Another application where transfer learning is prevalent is in generic visual recognition [7] where the results garnered using a pre-trained network are approximately the same as for the network trained from scratch on the training data set.

The other major technique used is Fine-Grained Classification. This is performed when the classes being detected are quite similar to one another, such as face or species recognition. Extensive work has been performed using fine-grained classification. Its base involves the detection of various similar objects like plants [4] and even cars [6] which makes this approach quite useful for our particular application especially given a limited data set and the similarity between the numerous classes.

For the major purpose of classification, apart from fine-tuning the renowned models such as Resnet, VGG and Densenet, we also looked into two other techniques. The first method we implemented was a variant of Ensemble Learning [8]. Ensemble Learning is the machine learning process through which better predictive performance can be obtained by combining multiple classification models into a single classifier [15]. In most conventional applications, the ensembles are used for multiple convolution layered networks [15], however we have implemented several different combinations which include Resnet, DenseNet and VGG networks, thus aiding us in giving better results.

A common approach to avoid nuisance factors while implementing fine-grained classification is to use variants of Convolutional Neural Networks by using handcrafted features manually [17].

This improves the classification accuracy however the manual process is tedious and unfeasible when targeting large datasets. To avoid this problem and get higher prediction accuracy, Bilinear Convolutional Neural Networks [17] were implemented. Bilinear Convolutional Neural Networks [23] are used to model two-factor variations i.e. the style and the content of the images by using the porduct of two low rank matrices to equip the model with a higher prediction ability. We used this in tandem with all the other techniques we implemented to evaluate its usefulness in our application.

4 Methods

In terms of methodology we developed and ran three series of tests which are described below. These tests consist of fine-tuning the Resnet, VGG, and Densenet class of models, an ensemble technique of fine-tuned models, and lastly a Bilinear Convolutional Neural Network (BCNN).

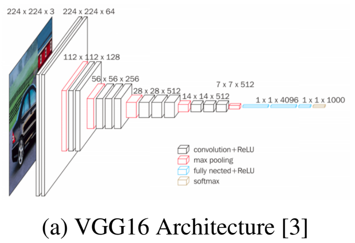
4.1 Fine-Tuning of Resnet, VGG, and Densenet

After fine-tuning the AlexNet architecture as our baseline model, we then moved on to more advanced networks to see how they would fair on our data set. Here we fine-tune three famous computer vision classes of models; Resnet, VGG, and DenseNet. For each architecture implemented, training is done in three ways. First the entire model is fine-tuned using stochastic gradient descent (SGD) with a learning rate of 0.001 and momentum of 0.9. Second, Adam is used as the optimizer again fine-tuning the entire model with a learning rate of 0.001, weight decay of zero, and beta values of 0.9 and 0.999 respectively. Finally, for the third variation only the final layer is fine-tuned leaving all convolutional filters as is based on their learned values from ImageNet. Here the optimizer used is either SGD or Adam based on whichever one led to better performance when fine-tuning the entire network. In all cases training was performed over 100 epochs or until over-fitting occurred.

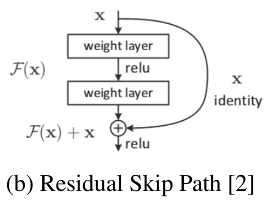
The first class of models fine-tuned was VGG [22] which was instrumental in showing that deeper CNNs played a critical role in better classification accuracy. More specifically this network uses max pooling, layers of 3x3 convolutional kernels of increasing filter count, and fully-connected layers at the end. In fine-tuning and testing the VGG class we use two specific depths, VGG-11 and VGG-16 (below). For each depth we train both the generic model as well the batch normalization version.

The second class of models fine-tuned was the Resnet [10] class. This architecture is credited with the introduction of the residual skip path (below) allowing for better training as a solution to the vanishing/exploding gradient problem. Thus, deeper networks can be trained resulting in increased accuracy. Here we train three specific variations, Resnet-18, Resnet-50, amd Resnet-152.

The last class of model we fine-tune is the Densenet [11] class. This architecture builds off the work from Resnet and advances the concept of the residual skip path. In fact, Densenet creates a connection between all layers in feed-forward fashion (below). "For each layer the feature-maps of all preceding layers are used as inputs, and its own feature-map is used as an input to all subsequent layers." This further reduces the vanishing/exploding gradient problem while strengthening feature propagation and allowing for even deeper networks. Here we fine-tune one Densenet architecture, Densenet-161.

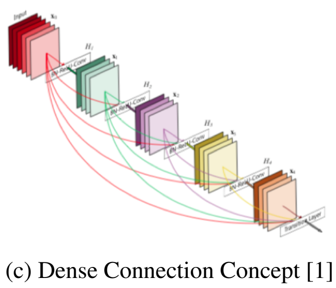


(a) VGG16 Architecture [3]



x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

(b) Residual Skip Path [2]


(c) Dense Connection Concept [1]

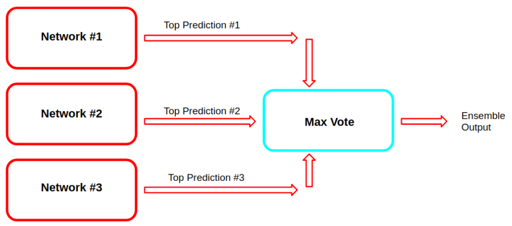
4.2 Ensemble of Fine-Tuned Models

After experimenting with fine-tuning VGG, Resnet, and Densenet we then construct various ensembles of these architectures to improve classification accuracy. Our inspiration behind this technique came from observing the specific classes each architecture was most likely to predict correctly / incorrectly. As will be shown in the results section, these inter-architecture class accuracy values vary from network type to network type. This is due not only to the randomness of training, but also because certain architectures seem to be more apt to distinguish the features corresponding to one class versus another. Thus an ensemble boosts final accuracy as models can support each other on poorly performing classes.

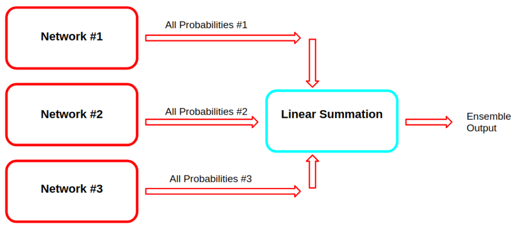
Here we experiment with a single ensemble size of three networks and two contrasting methods of combination. Our reasoning behind this size is two-fold. First we want the entire ensemble to fit into the 6GB of RAM on our GPU. This allows us to speed up inference times by avoiding the bottleneck of constantly reading architectures to memory. Additionally, this also allows for easier implementation of our max voting ensemble technique as we have an odd number of classifiers.

Building off of this, the two ensemble combination techniques we experiment with are max voting and linear summation. In max voting each network has a top predicted class. This technique works by choosing the most likely class based on the combination of top predictions. In other words, if two or three networks predict one specific class then this output is chosen. However, if all three classes vary, the class chosen is the one with the highest probability from the three models. In the case of linear summation, all class probabilities are added and the output is the top probability. This technique allows for increased output analysis as all classes are considered, not just the most likely one.

Our two ensemble techniques are as follows:



(a) Max Vote Ensemble Architecture



(b) Linear Summation Ensemble Architecture

4.3 Bilinear Convolutional Neural Network (BCNN)

The final architectural type we implement and experiment with is a Bilinear Convolutional Neural Network (BCNN) [16]. This network works by running images through two feature extractors in parallel, multiplying the outputs using outer product, sum average pooling across the image, passing this through the signed square root function, and finally running this through a fully-connected layer for classification. Typically the feature extractors used are CNNs with fully-connected layers removed. This makes sense as the convolutional and pooling sections of these models are the actual "feature extractor" whereas the fully-connected layers deal with classification.

Conceptually, computing this outer product allows both image features and locations to be considered. Basic CNNs work well due to the spacial locality of images, sections near each other often represent parts of an image and are combined into higher level features during processing. However, feature location can also play a role in classification. Take for example cars and say we have two classes of the same make and model but one year apart with the distinguishing feature being headlight location. A basic CNN may have difficulty in classifying these images as it's going to recognize the headlight structure and, as it is the same in both classes, could do a poor classification job. However, a BCNN is more apt to handle this problem as it will recognize the features and consider the feature location.

Digging deeper into the BCNN architecture and what we implement, the feature extractors used are the fine-tuned models of VGG, Resnet, and Densenet. These network structures are left unchanged except for all fully-connected layers being removed. After passing images through both feature extractors in parallel, outputs are multiplied using outer product and sum-average pooling is done across channels. For example, consider $A \in \mathbb{R}^{CxL}$ and $B \in \mathbb{R}^{CxJ}$ to be feature extractor outputs where C is the channel count and L and J are the feature spaces per channel ($L = Length_1 * Width_1; J = Length_2 * Width_2$). The sum average pooled outer product resulting from these two feature extractors would then be computed as follows:

$$Y = \frac{A^T B}{L * J}$$

As this is a matrix multiplication operation, $Y \in \mathbb{R}^{LxJ}$ and pooling occurs naturally across channels due to summation. Note that both L and J are scalar values and thus the division is simply diving the resulting matrix element-wise by a scalar, giving us the sum "average" pooling effect. The one constraint in regard to feature extractors is that the final filter count from convolutions must match up, the L variable. Fortunately, within specific families of models such as VGG and Resnet, final filter counts typically match.

After computing the sum average pooled outer products, results are then passed through the signed square root function element-wise. This function acts as a type of normalization.

$$y = sign(x)\sqrt{|x|}$$

Finally, values are sent into the fully-connected layer for classification. The overall structure of the BCNN architecture can be seen as follows:

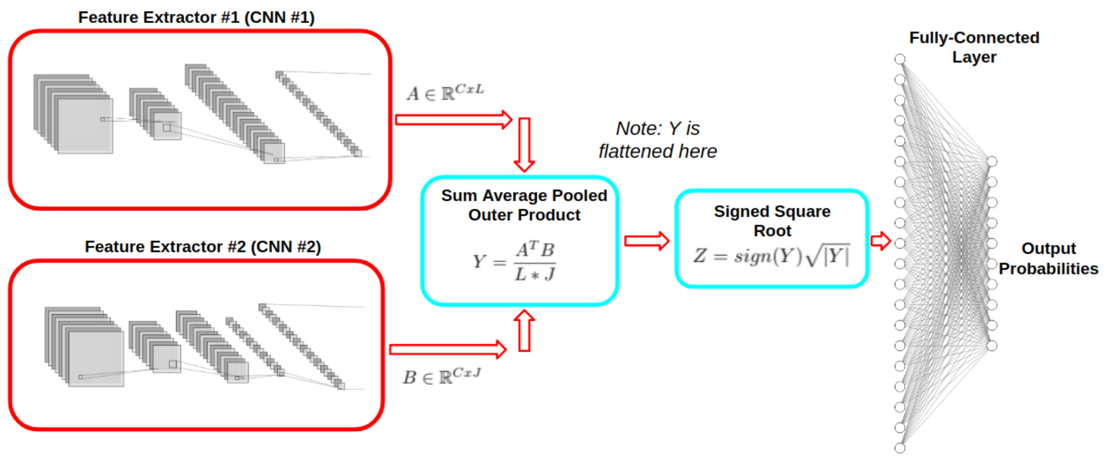

Feature Extractor #1 (CNN #1)
$A \in \mathbb{R}^{CxL}$
Feature Extractor #2 (CNN #2)
$B \in \mathbb{R}^{CxJ}$
Sum Average Pooled Outer Product
$Y = \frac{A^T B}{L * J}$
Note: Y is flattened here
Signed Square Root
$Z = sign(Y)\sqrt{|Y|}$
Fully-Connected Layer
Output Probabilities

Figure 4: Bilinear Convolutional Neural Network (BCNN)

To train BCNN models we use two techniques. In the first technique we use feature extractors fine-tuned from the earlier section of this project and randomly initialize the fully-connected layer. We then train this layer using SGD with a learning rate of 0.001 and momentum of 0.9 either for 100 epochs or until over-fitting. For this technique we stop the backward pass after the fully-connected layer and keep feature extractor parameters as is. In our second training variation we again use feature extractors developed earlier, randomly initialize the fully-connected layer, and use SGD to update the entire model. Training is done over 100 epochs or until over-fitting.

5 Results

5.1 Fine-Tuning of Resnet, VGG, and Densenet

Test set results of fine-tuning are shown below. For comparison we also include the results from our baseline AlexNet implementation. Again, the SGD optimizer uses a learning rate and momentum value of 0.001 and 0.9 respectively. For Adam the learning rate is 0.001, weight decay is zero, and beta values are 0.9 and 0.990. Accuracy values are as follows:

Model	Fine-Tuned Layers	Optimizer	Accuracy	Top 5 Accuracy
AlexNet	FC Only	SGD	24.35%	49.01%
AlexNet	Full Model	SGD	41.82%	69.05%
AlexNet	Full Model	Adam	35.53%	60.86%
VGG-11	FC Only	SGD	32.88%	62.22%

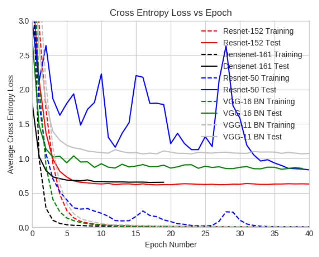
Model	Fine-Tuned Layers	Optimizer	Accuracy	Top 5 Accuracy
VGG-11	Full Model	SGD	69.29%	90.55%
VGG-11	Full Model	Adam	48.35%	78.85%
VGG-11 BN	FC Only	SGD	39.93%	68.51%
VGG-11 BN	**Full Model**	**SGD**	**75.44%**	**93.05%**
VGG-11 BN	Full Model	Adam	55.39%	78.44%
VGG-16	FC Only	SGD	34.04%	63.98%
VGG-16	Full Model	SGD	75.03%	92.97%
VGG-16	Full Model	Adam	51.66%	82.02%
VGG-16 BN	FC Only	SGD	39.32%	68.47%
VGG-16 BN	**Full Model**	**SGD**	**81.99%**	**95.56%**
VGG-16 BN	Full Model	Adam	61.04%	85.23%
Resnet-18	FC Only	Adam	36.50%	65.14%
Resnet-18	Full Model	SGD	64.26%	87.80%
Resnet-18	Full Model	Adam	75.51%	93.01%
Resnet-50	FC Only	Adam	41.47%	68.90%
Resnet-50	Full Model	SGD	74.98%	93.20%
Resnet-50	**Full Model**	**Adam**	**82.88%**	**95.86%**
Resnet-152	FC Only	SGD	44.73%	73.05%
Resnet-152	**Full Model**	**SGD**	**84.18%**	**96.38%**
Resnet-152	**Full Model**	**SGD (1)**	**83.55%**	**96.73%**
Resnet-152	Full Model	Adam	78.93%	94.53%
Densenet-161	**Full Model**	**SGD**	**83.70%**	**96.05%**
Densenet-161	Full Model	Adam	80.59%	94.89%

*Note: The models in **BOLD** are later used in ensemble and BCNN analysis*

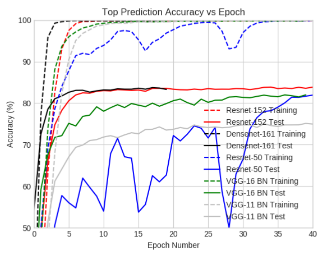
Note: (1) indicates weight decay=0.001

As expected, fine-tuning an entire model versus only the final layer leads to significantly better results. We can also see the power and benefit from deeper architectures as accuracy increases with increasing model depth. This is especially prevalent within the Resnet and Densenet classes as their residual / dense connections allow for models over a hundred layers to be trained. One final takeaway is that batch normalization leads to significantly better results for the VGG class. This technique stabilizes training and increases final accuracy by over 5-6%.

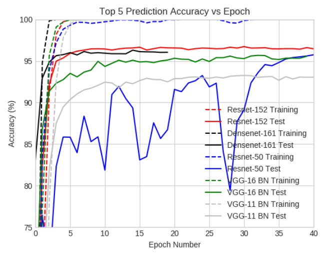
Overall, fine-tuning these more advanced CNNs significantly outperformed the baseline AlexNet implementation. This was expected as the VGG, Resnet, and Densenet classes are deeper architectures better suited for image recognition. Our best performing models were Resnet-152 and Densenet-161 with SGD training. These models reached top prediction accuracy of 84.14% and 83.70% and top five prediction accuracy of 96.38% and 96.05% respectively. A plot of the loss, accuracy, and top five accuracy per epoch for some of our better performing models are as follows:



(a) Model Loss



(b) Model Accuracy



(c) Model Top 5 Accuracy

5.2 Ensemble of Fine-Tuned Models

The next series of tests implemented consist of various ensemble combinations of the fine-tuned networks. The motivation behind this technique stemmed from different architectures performing better on certain classes. For example, consider the results below where we look at the most/least accurate classes for specific architectures and the percentage of test points each correctly predicts:

Note: Label names shortened for space consideration

Densenet-161	**Resnet-152**	**Resnet-50**
FIAT 500 Abarth 2012: 100.00%	Chrysler PT Cruiser 2008: 100.00%	Cadillac CTS-V 2012 100.00%
Jeep Wrangler 2012: 100.00%	FIAT 500 Abarth 2012: 100.00%	FIAT 500 Abarth 2012: 100.00%
Volkswagen Beetle 2012: 100.00%	Isuzu Ascender 2008: 100.00%	Ford F-450 2012: 100.00%
Volkswagen Golf 1991: 97.83%	Jeep Wrangler 2012: 100.00%	Scion xD 2012: 100.00%
Cadillac Escalade 2007: 97.72%	Volkswagen Golf 1991: 97.83%	Dodge Caravan 1997: 97.67%
...	...	...
Audi TT Hatchback 2011: 52.50%	Aston Martin V8 2012: 57.78%	Chevrolet Silverado 2012: 57.50%
Chevrolet Express 2007: 51.43%	Audi S5 Coupe 2012: 57.14%	Dodge Caliber 2007: 54.76%
Audi TTS Coupe 2012: 50.00%	Chevrolet Express 2007: 54.29%	Audi TT Hatchback 2011: 52.50%
BMW 6 Series 2007: 50.00%	Audi TT Hatchback 2011: 45.00%	Chevrolet Express 2007: 48.57%
Audi V8 Sedan 1994: 48.83%	Chevrolet Cargo 2007: 41.38%	Chevrolet Cargo 2007: 44.83%

As can be seen, some architectures perform significantly better on specific classes compared to others. For example, the second least accurate class from Densenet-161 is *BMW 6 Series Convertible 2007* where accuracy is 50.00%. However, this class is **NOT** within the least ten accurate classes for either Resnet-152 or Resnet-50 with both networks able to correctly classify these images at over 61.50%.

Based on this analysis we experiment with two ensemble techniques, max voting and linear summation, as well as different combinations of fine-tuned networks. Here are our results:

Ensemble Combination	**Network 1**	**Network 2**	**Network 3**	**Accuracy**	**Top 5 Accuracy**
Max Vote	VGG-16 BN	Resnet-50	Resnet-152	87.14%	98.51%
Linear Summation	VGG-16 BN	Resnet-50	Resnet-152	89.72%	98.51%
Linear Summation	Resnet-50	Resnet-152	Resnet-152(2)	89.01%	98.11%
Linear Summation	Densenet-161	Resnet-152	Resnet-152(2)	88.91%	98.16%
Linear Summation	**Densenet-161**	**Resnet-50**	**Resnet-152**	**90.19%**	**98.40%**

Note: (2) indicates the weight decay trained version of Resnet-152

With a linear summation ensemble consisting of Densenet-161, Resnet-50, and Resnet-152 we were able to max out our classification accuracy at 90.19% with a top five accuracy of 98.40%. This improves upon the accuracy of our best single model by over 6% showing the power of ensemble techniques. This method allows three networks to support each other particularly on classes that a single model performs poorly on. Additionally, it was clear that linear summation was superior to max voting and thus this technique was only implemented for one ensemble.

5.3 Bilinear Convolutional Neural Network (BCNN)

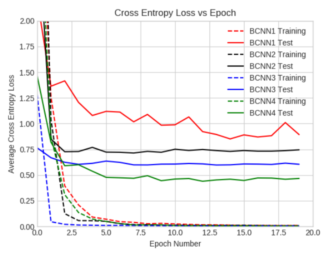
The final series of experiments implemented use the Bilinear Convolutional Neural Network (BCNN) architecture. First we train the fully-connected layer only and leave feature extractors as is. Feature extractors used are CNNs trained in earlier experimentation. Unfortunately, this technique failed to perform well and surpass the accuracy of our best performing single model or ensemble.

The next step was to allow the entire network to be trained. Here SGD is again used and we experiment with a variety of network combinations as feature extractors as well as two image resolutions, 224x224 and 448x448. Our thinking behind working with the larger resolution was that less image compression would be done potentially leading to better classification results. BCNN accuracy results follow:

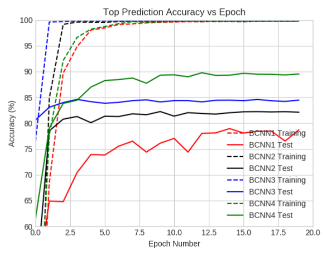
Name	**Image Resolution**	**Network 1**	**Network 2**	**Accuracy**	**Top 5 Accuracy**
BCNN1	224x224	VGG-11	VGG-16	78.96%	93.87%
BCNN2	224x224	Resnet-152	Resnet-50	82.40%	95.61%
BCNN3	224x224	VGG-11 BN	VGG-16 BN	84.82%	96.43%
BCNN4	**448x448**	**VGG-11 BN**	**VGG-16 BN**	**89.78%**	**98.15%**

Based on these results it appears that the VGG class works better as a feature extractor compared to Resnet. By using VGG we were able to increase accuracy by about 2-3% as compared to a single VGG network. Unfortunately we failed to see this same improvement with Resnet.

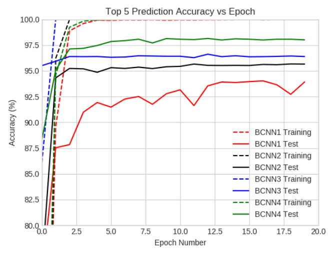
Finally, it appears that using the larger image resolution does play a role in improving accuracy. Using the same original feature extractors and doubling resolution resulted in an improvement in classification accuracy by nearly 5%. This was our best performing model at an accuracy of 89.78% and top 5 accuracy of 98.15% coming close to performing as well as our optimal ensemble method. Below are the training curves documenting the loss, accuracy, and top 5 accuracy of networks above:



(a) BCNN Loss



(b) BCNN Accuracy



(c) BCNN Top 5 Accuracy

6 Discussion and Analysis

We can deduce several key observations after implementing the aforementioned models. Firstly, the results conclude definitively that the final accuracy we achieve on the test set is dependent on the model we used and the optimizer utilized. Theoretically, the classification accuracy should remain invariant to the optimizer as the model has the same potential representational space. However, in our case, the primary reason for varying accuracy is that the hyper-parameters being used are non-optimal which result in the accuracy being dependent on the optimization scheme. If we had been able to explore a greater number of hyper-parameters in the representational space, we expect that the accuracy would remain almost the same regardless of the optimizer used.

Another key observation we can deduce after using the Bilinear Convolutional Neural Network is that the model could have performed better. BCNN is a powerful technique and still outperforms each of the single models by a margin but it could be more powerful. One of the key reasons why the model did not perform as well as it could have is due to non-optimal hyper-parameters. Whenever a machine or deep learning model is created, there are design choices which must be specified before-hand for the model. If we do not know of the optimal model architecture for the model in the first place, we would have to search extensively over all possible parameters to find the optimal values. However, in the case of BCNN, the model is very computationally expensive and thus takes a substantially large amount of time to train. This resulted in extensive hyper-parameter exploration which proved to be very difficult and thus is one of the reasons why BCNN did not perform as well as expected.

In the case of BCNN, another interesting remark is the effect of image resolution on the classification accuracy. When we double the image resolution keeping the original feature descriptors, the prediction accuracy increased by 5-6%. This shows that using larger image resolutions for single fine-tuned networks could also result in considerable accuracy increase. One difference between our experiments as compared to several other papers is the fact that the training data set was not altered. In many of the previous works, the training set is enhanced by increasing its size in different ways such as adjusting lighting, flipping images, etc. Our models work on the original unaltered data-sets while still resulting in a comparable classification accuracy.

Finally, we compare our results to some of the previous works and look at how well our models perform. In comparison with the work of Xiao Lu and Tian Xia [19], our maximum classification accuracy of 90.19% using an ensemble classifier was greater than their predicted accuracy of 80.00% using the GoogLeNet model full-trained. In comparison to the work of Goslin [9] et. al which achieved a maximum accuracy of 82.7%, our model performed better. However, Krause et. al [13] predicted with an accuracy of 92.6% using no parts annotation which was better performing.

References

[1] Densely connected convolutional networks | arthur douillard. `https://arthurdouillard.com/post/densenet/`. (Accessed on 05/04/2019).

[2] Residual blocks—building blocks of resnet – towards data science. `https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec`. (Accessed on 05/04/2019).

[3] Vgg16 - convolutional network for classification and detection. `https://neurohive.io/en/popular-networks/vgg16/`. (Accessed on 05/01/2019).

[4] Anelia Angelova and Shenghuo Zhu. Efficient object detection and segmentation for fine-grained recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 811–818, 2013.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[6] Louka Dlagnekov and Serge J Belongie. *Recognizing cars*. Department of Computer Science and Engineering, University of California . . . , 2005.

[7] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[8] Muhammad Moazam Fraz, Paolo Remagnino, Andreas Hoppe, Bunyarit Uyyanonvara, Alicja R Rudnicka, Christopher G Owen, and Sarah A Barman. An ensemble classification-based approach applied to retinal blood vessel segmentation. *IEEE Transactions on Biomedical Engineering*, 59(9):2538–2548, 2012.

[9] Philippe-Henri Gosselin, Naila Murray, Hervé Jégou, and Florent Perronnin. Revisiting the fisher vector for fine-grained classification. *Pattern recognition letters*, 49:92–98, 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[11] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL `http://arxiv.org/abs/1608.06993`.

[12] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[13] Jonathan Krause, Hailin Jin, Jianchao Yang, and Li Fei-Fei. Fine-grained recognition without part annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5546–5555, 2015.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[15] Ashnil Kumar, Jinman Kim, David Lyndon, Michael Fulham, and Dagan Feng. An ensemble of fine-tuned convolutional neural networks for medical image classification. *IEEE journal of biomedical and health informatics*, 21(1):31–40, 2017.

[16] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1449–1457, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.170. URL `http://dx.doi.org/10.1109/ICCV.2015.170`.

[17] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear cnn models for fine-grained visual recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1449–1457, 2015.

[18] Derrick Liu and Yushi Wang. Monza: image classification of vehicle make and model using convolutional neural networks and transfer learning, 2017.

[19] Xiao Liu, Tian Xia, Jiang Wang, Yi Yang, Feng Zhou, and Yuanqing Lin. Fully convolutional attention networks for fine-grained recognition. *arXiv preprint arXiv:1603.06765*, 2016.

[20] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[23] Joshua B Tenenbaum and William T Freeman. Separating style and content with bilinear models. *Neural computation*, 12(6):1247–1283, 2000.